ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels *

Daniel Genkin
Technion and
Tel Aviv University
danielg3@cs.technion.ac.il

Lev Pachmanov
Tel Aviv University
levp@tau.ac.il

Itamar Pipman
Tel Aviv University
itamarpi@tau.ac.il

Eran Tromer
Tel Aviv University
tromer@tau.ac.il

Yuval Yarom
The University of Adelaide and
Data61, CSIRO
yval@cs.adelaide.edu.au

ABSTRACT

We show that elliptic-curve cryptography implementations on mobile devices are vulnerable to electromagnetic and power side-channel attacks. We demonstrate full extraction of ECDSA secret signing keys from OpenSSL and CoreBitcoin running on iOS devices, and partial key leakage from OpenSSL running on Android and from iOS's CommonCrypto. These non-intrusive attacks use a simple magnetic probe placed in proximity to the device, or a power probe on the phone's USB cable. They use a bandwidth of merely a few hundred kHz, and can be performed cheaply using an audio card and an improvised magnetic probe.

1 Introduction

1.1 Overview

Side channel analysis, exploiting unintentional and abstraction-defying information leakage from physical computation devices, has been used to break numerous cryptographic implementations (see [9,40,42] and the references therein). While traditional side channel research has mainly focused on small embedded devices such as smartcards, RFID tags, FPGAs and microcontrollers, recent works also study the vulnerability of complex PC-class computers (laptop, desktop and server) to physical key-extraction attacks [29–32, 59].

In this paper we study vulnerability to side-channel key extraction in another class of complex devices: mobile devices (smartphones and tablet computers). This prospect is already supported by some recent results. Using invasive access to the device, it is possible to acquire electromagnetic and power measurements with very high fidelity in terms of bandwidth, noise and spatial locality. Such invasive access has been used for key extraction attacks on intentionally-naive RSA implementations [33, 45]. A non-invasive attack was shown by Kenworthy and Rohatgi [4,38] on BouncyCastle's RSA implementation running on a smartphone. All of these attacks used expensive lab-grade equipment, such as oscilloscopes, for their measurements.

This paper focuses, instead, on the Elliptic Curve Digital Signature Algorithm (ECDSA) [46], a very popular signature scheme that is especially pertinent and critical in mobile devices due to its use in mobile payment apps such as Bitcoin wallets and Apple Pay. Attacking ECDSA raises new challenges:

- ECDSA signatures are computed faster than RSA, and thus the attacker gets less physical information at a given sampling rate. Increasing the sampling rate increases costs and runs into frequency-limited physical effects.
- More fundamentally, ECDSA signatures are randomized. When attacking deterministic operations, such as RSA decryption, attackers can rely on triggering numerous identical decryptions and then aggregating their recorded traces in order to improve signal-to-noise ratio and cope with transient events such as interrupts. But with ECDSA, one has to make deductions from individual traces that are noisy and frequently interrupted.

We raise the following questions:

1. How vulnerable are implementations of ECDSA, running on mobile phones, to physical side channel attacks?
2. Are these vulnerabilities common across different implementations and across different phone models?
3. What physical channels can be used for the attacks?
4. How expensive are such attacks, both in terms of complexity and in terms of financial outlay? Can they be conducted with concealed, portable equipment? Do they require high-grade lab equipment or can they be implemented using cheap, over-the-shelf equipment?

A concurrent and independent work of Belgarric et al. [14] provides a valuable insight on some of these questions, demonstrating full key extraction from BouncyCastle's ECDSA implementation on a phone. That attack used an electromagnetic probe placed invasively inside the open case of a phone. It relied on triggering measurement via the USB interface, and (even though essentially relying on low-frequency sig-

*The authors thank Noam Nissan for programming and lab support during the course of this research.

CCS'16 October 24-28, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2978353


Library	Version	Platform	Result
OpenSSL	1.0.x 1.1.x	Android	Scalar dependent leakage (Section 3.3).
OpenSSL	1.0.x 1.1.x	iOS	Key extraction (Section 3.2).
Common-Crypto	7.1.2–8.3	iOS	Scalar dependent leakage (Section 3.3).
CoreBitcoin	Commit `81762ae3`	iOS	Key extraction (Section 3.3).

Table 1: Summary of our attack results.

nals) used an expensive oscilloscope. This leaves unexplored much of the space posed by the aforementioned questions.

1.2 Our Results

In this paper we demonstrate the first side channel attack on Elliptic Curve Cryptography (ECC) running on a smartphone which *simultaneously* achieves the following:

1. **Real-World Implementations.** We attacked the ECDSA implementation of OpenSSL running on iOS devices (iPhone and iPad) as well as Android devices. In particular, we attacked the CoreBitcoin library, based on OpenSSL, which is used by popular Bitcoin wallets on iOS devices. We also attacked the built-in ECDSA implementation of iOS's CommonCrypto library.
2. **Non-Invasive.** The demonstrated attacks are non invasive and can be conducted by merely placing a magnetic probe in the proximity of the device, or using a power tap on its USB charging cable. The attack does not require opening the device's case and does not utilize any software or hardware in order to trigger the measuring equipment. See Figures 1 and 9.[1]
3. **Cheap EM and Power Analysis.** Our attack utilizes physical emanations (electromagnetic or power) at frequencies below 200 kHz, which is well below the GHz-scale processor clock speed. Consequentially, our attack can acquire secret-key information using cheap, compact and readily available equipment, such as sound cards and improvised probes.

In some cases (e.g., CoreBitcoin on iPhone devices), we demonstrated full key extraction. It was impractical to do so for all combinations of target software, target hardware and acquisition hardware, but for numerous such combinations we found clear leakage of key material suggesting feasibility of full key extraction, as discussed in Sections 3 and 4. See Table 1 for a summary of our results.

We achieve the above using new techniques for enhancing the measured side-channel signal in the presence of noise generated by the device's internal components. While typical techniques for overcoming measurement noise involve averaging the signal obtained from several secret-key operations, this is not applicable to ECDSA since the nonce is generated afresh for every signature. Instead, we present techniques for enhancing the signal present in a single trace, without relying on additional information from other traces.

[1]While we do not require any specific triggering software, the attack does require the target to repeatedly perform ECDSA signing operations. Several such scenarios exist, see Section 1.3.

1.3 Targeted Software and Hardware

Hardware. We target mobile devices such as tablets and phones. We measured numerous devices of various models. Many devices exhibit key-dependent leakage (see Figure 3). All the devices were in their default configuration and we did not disable any background services and notifications. WiFi was on and connected and bluetooth was off. All phones were without a simcard installed.[2] In the sequel, unless stated otherwise, the experiments were performed on Apple iPhone 3GS which exhibited a particularly clear signal.

Software. We target popular ECDSA implementations running on various mobile devices. More specifically, we target the following implementations:

1. The ECDSA implementation of **OpenSSL** (v1.0.1m), a ubiquitous cryptographic library, running on iOS and Android devices.[3] The underlying Elliptic Curve (EC) scalar multiplication algorithm is wNAF with $w = 3$.
2. The built-in ECDSA implementation of Apple's **CommonCrypto** library, which is a part of iOS. We targeted iOS versions 7.1.2 through 8.3, whose underlying EC multiplication algorithm is wNAF with $w = 1$.
3. **CoreBitcoin** [2], a popular cryptographic library for iOS used by many Bitcoin clients (including ArcBit, BitStore, BitWallet, Mycelium and Yellet). CoreBitcoin implements deterministic ECDSA [52], using OpenSSL for the underlying EC multiplication.

Attack Scenario. Our attacks require side-channel measurements while the victim performs multiple ECDSA signing operations. Signing multiple messages under the same key is common when the key is fixed by a public key infrastructure or a PGP "web of trust". It is also necessary for Bitcoin micropayment channels [3, 56], which allow making lightweight out-of-blockchain automated payments for an ongoing service. Since each such micropayment requires an ECDSA signature this will cause frequent and automated signatures under the same key.

Disclosure and Status. Practicing responsible disclosure, we have worked with the vendors of all targeted software to convey our findings and coordinate response, prior to public disclosure. See Appendix A for the current status of targeted software, including newer versions.

1.4 Related Work

Physical Attacks on ECC on Small Devices. For small devices (smartcards, RFID tags, FPGAs and microcontrollers), side-channel attacks have been demonstrated on numerous cryptographic implementations, using various channels, and in particular the EM channel starting with [6, 28, 54]. See [9, 40, 42] and the references therein. In particular, physical key-extraction attacks were shown on many ECC implementations on small devices, starting with Coron [24]; see the surveys [25, 26] and the references therein. However these techniques either utilize subtle physical effects

[2]The phones are SIM-locked to a foreign carrier and we do not have the appropriate sim cards.

[3]We used OpenSSL compiled with its default options. In particular, the "`enable-ec_nistp_64_gcc_128`" option, which enables a constant-time implementation for some curves [37], is disabled by default and works only on 64-bit x86 (32-bit processors are unsupported, and the built-in tests fail on 64-bit ARM).

(a) Top view. The target (top right) is measured by the improvised probe (taped to the underside of a glass table). The signal is captured by a Tracker Pre sound card connected to a laptop (under the table).

(b) Improvised probe (view from under the glass table).

Figure 1: Mounting a cheap EM attack on an iPhone 4 using an improvised EM probe.

which are only visible at bandwidths comparable to the device's clock rate, attack naive implementations (such as the double-and-sometimes-add algorithm), or utilize a chosen ciphertext in order to deduce additional information about the algorithm's secret internal state.

All of the above approaches have significant drawbacks in the case of a non-naive implementation of ECDSA running on a high-speed smartphone. Non-invasively recording clock-rate signals from a smartphone running a multi GHz-scale CPU is difficult, often requiring expensive, cumbersome, and delicate lab equipment. Chosen-input attacks are usually inapplicable to signature schemes, since the inputs are processed through a cryptographic hash function.

Key-Extraction Side-Channel Attacks on Phones. High bandwidth electromagnetic attacks (sampling at clock-rate speeds) on symmetric ciphers were demonstrated by Aboulkassimi et al. [5] on Java-based feature-phones. Attacks at clock-rate frequencies on public key cryptography were also recently demonstrated by Goller and Sigl [33] on Android smartphones running a naive square-and-sometimes-multiply RSA, with the phone's shielding plate often removed. Lower-frequency attacks on smartphones executing naive implementations of square-and-sometimes-multiply RSA as well as double-and-sometimes-add ECC were also demonstrated by [45] with the phone battery cover opened, battery removed and the probe positioned directly over the leaky component. A non-invasive low-frequency attack was demonstrated by Kenworthy and Rohatgi [38] against naive square-and-sometimes-multiply RSA. These attacks were also later extended to RSA windowed exponentiation as used in BouncyCastle [4]. Finally, measuring at clock-rate frequencies, the work of Kenworthy and Rohatgi [38] also presented an attack on a naive and self-written double-and-sometimes-add ECC, which is known to have side-channel weakness.

Belgarric et al. [14], in a concurrent and independent work (as acknowledged by them [15]), presented an invasive low-frequency attack on the ECDSA implementation of Android's BouncyCastle library, running on a smartphone, using a magnetic probe placed inside the (opened) phone. It used a bandwidth of 50 kHz, measured by an oscilloscope. The oscilloscope was triggered, via a self-written triggering software installed on the phone. The software sends a trigger signal via phone's USB port (connected to oscilloscope's triggering port) and then immediately invokes the BouncyCastle signing function. The acquisition and analysis of the signal were done by manual observation of the DOUBLE and ADD operations in the (hundreds of) traces.

Software side-channel attacks, utilizing cache contentions, were demonstrated on ARM devices, showing partial extraction of an AES key [41]. These require attacker's code to run on the device.

Key Extraction Attacks on ARM Development Boards. In recent works, DPA-style attacks were demonstrated on ARM-type devices. Balasch et al. [13] demonstrated a clock-rate attack on AES running on a BeagleBone Black ARM development board. A similar attack on the same device at much lower frequencies was demonstrated by Galea et al. [27]. While these results demonstrate the possibility of attacking symmetric key encryption running on complex devices, both attacks were highly invasive with the probe physically glued to the leaky component. The task of demonstrating a non-invasive attack on symmetric key encryption running on a real smartphone remains open.

Key-Extraction Side-Channel Attacks on PCs. Physical key-extraction side-channels were exploited for extracting keys from RSA, ElGamal and ECDH implementations, using the acoustic, chassis-potential and electromagnetic channels [29–32]. Software key extraction attacks were demonstrated using timing differences [20,21], and cache contention [17, 49, 50] and applied to ECDSA [8, 16, 19, 51].

2 Cryptanalysis

2.1 Preliminaries

ECDSA. We start by describing the Elliptic-Curve Digital Signature Algorithm scheme (ECDSA). Given a generator $\mathbb{G}$ of an elliptic curve group of order n, key generation consists of selecting a random integer $1 \leq d \leq n-1$ and computing $\mathbb{Q} = [d]\mathbb{G}$. (Here and onward, we use additive group notation, and $[d]\mathbb{G}$ denotes scalar-by-point multiplication.) The (secret) signing key is d and the (public) verification key is $\mathbb{Q}$. Signing of a message m is done as follows: hash m under a designated hash function and convert the first $\lceil \log_2 n \rceil$ bits of the digest into an integer z; generate a random nonce $1 \leq k \leq n-1$; compute the curve point $(x, y) = [k]\mathbb{G}$ using a scalar-by-point multiplication; compute $r = x \bmod n$ and $s = k^{-1}(z + r \cdot d) \bmod n$; output the signature (r, s). (In case $r = 0$ or $s = 0$, repeat the signature operation using a fresh random k.) Verifying a signature (r, s) on m is done by computing z as above, computing $w = s^{-1} \bmod n$, $u_1 = zw \bmod n$, $u_2 = rw \bmod n$, $(x, y) = [u_1]\mathbb{G} + [u_2]\mathbb{Q}$ and then checking that $x \equiv r \pmod n$.

Low *s*-value ECDSA. ECDSA signatures are malleable in the sense that given a message m and a signature pair (r, s), it is possible to generate an additional valid signature for m $(r', s') \neq (r, s)$ by setting $r' = r$ and $s' = -s \bmod n$. This property is problematic for Bitcoin clients which use the hashing of (r, s) in order to identify matching signatures [10, 58]. A common solution to the above problem used by many Bitcoin clients is to require that $s \leq n/2$. That is, in the case that the signing process (described above) generates a pair (r, s) where $s > n/2$ for some message m, the signature routine outputs $(r, -s \bmod n)$ as the signature of m [58].

Notation. In the sequel, for an integer a, we denote by $|a|$ is absolute value, by $\lfloor a \rfloor_n$ the result of reducing a modulo n into the range $[0, \cdots, n-1]$ and by $|a|_n$ the result of reducing a modulo n into the range $[-n/2, \cdots, n/2)$.

2.2 Scalar-by-Point Multiplication

The main operation performed during a ECDSA signing is the elliptic curve scalar-by-point multiplication. The *w-ary non-adjacent form (wNAF)* method (Algorithm 1) is one of the commonly-used algorithms for implementing scalar-by-point multiplication. wNAF is used for multiplication in curves over prime size fields, including many of the NIST P-curves and the Bitcoin curve secp256k1, in several cryptographic libraries, such as OpenSSL, CoreBitcoin, Apple's CommonCrypto and BouncyCastle. The algorithm is so named for representing the scalar k using the wNAF representation which we now discuss.

The *non adjacent form* [55] is a generalization of the binary representation of integers, allowing for three possible digits, -1, 0, and 1, referred to as *NAF digits*, and requiring that every pair of non-zero digits is separated by at least one zero digit. For example, the 4-digit NAF representation of 7 is $(1,0,0,-1)$ compared to its binary representation (0,1,1,1). The main advantage of using a NAF representation is that it reduces the expected number of non-zero digits from about 1/2 for the binary representation to about 1/3. Since every non zero digit in the representation of k leads to a point addition operation, representing k in NAF form reduces the number of point addition operations performed during the scalar-by-point multiplication operation.

Algorithm 1 wNAF scalar-by-point multiplication operation (simplified).

Input: A positive scalar k and an elliptic-curve point $\mathbb{P}$, where $k_{g-1} \cdots k_0$ is the wNAF representation of k, that is $k = \sum_{i=0}^{g-1} 2^i \cdot k_i$, $k_i \in \{-2^w + 1, \cdots, 2^w - 1\}$, k_i is odd or zero, and $k_{g-1} \neq 0$.
Output: $[k]\mathbb{P}$.

```
1:  procedure POINT_MUL(k, P)
2:      Q_1 ← P ; Q_{-1} ← [-1]P
3:      for i ← 1 to 2^{w-1} − 1 do
4:          Q_{2i+1} ← Q_{2i-1} + [2]P
5:          Q_{-2i-1} ← [-1]Q_{2i+1}
6:      A ← Q_{k_{h-1}}
7:      for i ← g − 2 to 0 do
8:          A ← [2]A
9:          if k_i ≠ 0 then
10:             A ← A + Q_{k_i}
11:     return A
```

The wNAF representation generalizes this by allowing odd digits from $\{-2^w + 1, \cdots, 2^w - 1\}$ as well as zero digits.

2.3 Attack Algorithm

Let *DA-sequence* denote the sequence of DOUBLE and ADD operations performed in lines 8 and 10 of Algorithm 1. Notice that by observing the DA-sequence performed by Algorithm 1 it is possible to deduce all the locations of the non-zero valued wNAF digits of the nonce k. However, since the DA-sequence only discloses the positions of the non-zero digits but not their values, recovering the DA-sequence alone is not enough for achieving key extraction.

Cryptanalytic Approach. Nguyen and Shparlinski [48] describe a theoretical attack for combining partial information on the bits of multiple nonces in order to recover the secret key d. Benger et al. [16] later apply the attack to the DA-sequences of OpenSSL's implementation of ECDSA, as leaked through a cache channel on a PC. In this section we extend these techniques for handling low s-value ECDSA commonly used by Bitcoin clients.

In our approach, following [16, 18, 48, 51], the partial information collected from each (suitable) signing operation is summarized in a matrix. The secret value is then extracted by solving the *Closest Vector Problem* (CVP) on the corresponding lattice, i.e., by finding an integer linear combination of the matrix rows that is close a to a target vector. Details follow.

Closest Vector Problem. An input of a CVP consists of a matrix (lattice basis) B and a target vector $\mathbf{u}$. The output is an integer vector $\mathbf{x}$ such that the ℓ_2-norm of the vector $\mathbf{x}B - \mathbf{u}$ is minimal, i.e., the lattice vector $\mathbf{x}B$ is the closest to $\mathbf{u}$. While the CVP problem is believed hard in general and the best algorithms are exponential in the dimension of B (in the worst case), many heuristic CVP solvers exists [47]. In this work, we utilize the fplll solver [7] running on a PC (3.4 GHz, 6 cores, 64 GB of RAM).

Attacking Low *s*-value ECDSA. Let d be an ECDSA signing key and $\mathbb{G}$ be a generator of an elliptic curve of order n. Assume we have a dataset of m ECDSA signatures where for each signature i we are given the hashed message z_i and the signature (r_i, s_i), where s_i is the low s-value, i.e.

$s \leq n/2$. First, we notice that for all i it holds that

$$z_i + d \cdot r_i \equiv s_i k_i \pmod{n} \quad \text{or} \quad z_i + d \cdot r_i \equiv -s_i k_i \pmod{n}. \tag{1}$$

Notice that, without knowing k_i, we do not know which of the above cases holds. (This depends on whether $k^{-1}(z + r \cdot d) \bmod n$ is larger than $n/2$ or not.)

Assume that, for each signature i in this dataset, we have learned (through side-channel leakage) that the l_i least significant wNAF digits of k_i are zero. We first note that this also implies that the l_i least significant bits of k_i are zeros or, equivalently, that $k_i = 2^{l_i} \cdot b_i$ for some $b_i \leq n/2^{l_i}$. Expanding and rearranging Equation 1 we obtain that for all i it holds that

$$\begin{aligned} z_i \cdot s_i^{-1} \cdot 2^{-l_i} + d \cdot r_i \cdot s_i^{-1} \cdot 2^{-l_i} &\equiv b_i \pmod{n} \quad \text{or} \\ z_i \cdot s_i^{-1} \cdot 2^{-l_i} + d \cdot r_i \cdot s_i^{-1} \cdot 2^{-l_i} &\equiv -b_i \pmod{n}. \end{aligned} \tag{2}$$

Next, define $t_i = \lfloor r_i \cdot s_i^{-1} \cdot 2^{-l_i} \rfloor_n$, $u_i = \lfloor z_i \cdot s_i^{-1} \cdot 2^{-l_i} \rfloor_n$ and $\nu_i = |dt_i - u_i|_n$. From Equation 2 we have that either $\nu_i \equiv b_i \pmod{n}$ or that $\nu_i \equiv -b_i \pmod{n}$. Finally, since $b_i \leq n/2^{l_i} \leq n/2$ and $|\nu_i| \leq n/2$, we obtain:

$$|\nu_i| = b_i \leq n/2^{l_i}. \tag{3}$$

Notice that $|\nu_i|$ is smaller by a factor of $2^{l_i - 1}$ than a random element in $\mathbb{Z}_n$. Utilizing this fact, following the approach of [16, 48], we now convert our dataset into a closest vector lattice problem.

CVP Attack. Consider the lattice $\mathcal{L}(B)$ over $\mathbb{R}^{m+1}$ generated by the rows of the following matrix:

$$B = \begin{pmatrix} 2^{l_1} \cdot n & & & \\ & \ddots & & \\ & & 2^{l_m} \cdot n & \\ 2^{l_1} \cdot t_1 & \cdots & 2^{l_1} \cdot t_m & 1 \end{pmatrix}.$$

Define the vector $\mathbf{u} = (2^{l_1} \cdot u_1, \cdots, 2^{l_m} \cdot u_m, 0)$. Notice that both the matrix B and the vector $\mathbf{u}$ can be computed from the public information z_i, s_i and the leakage l_i for $1 \leq i \leq m$. We now claim that the solution to the closest vector problem defined by $\mathcal{L}(B)$ and $\mathbf{u}$ reveals the secret key d.

Indeed, Equation 3 implies the existence of integers $(\lambda_1, \cdots, \lambda_m)$ such that for the vectors $\mathbf{x} = (\lambda_1, \cdots, \lambda_m, d)$ and $\mathbf{y} = (2^{l_1} \cdot \nu_1, \cdots, 2^{l_m} \cdot \nu_m, d)$ we have $\mathbf{x}B - \mathbf{u} = \mathbf{y}$. Next, notice that the ℓ_2-norm of $\mathbf{y}$ is about $n \cdot \sqrt{d+1}$ whereas the determinant of $\mathcal{L}(B)$ is $2^{m + \sum l_i} \cdot n^m$. Thus, we obtain that the lattice vector $\mathbf{x}B$ is heuristically closest to the vector $\mathbf{u}$. By solving the CVP problem on inputs $(B, \mathbf{u})$ we obtain the vector $\mathbf{x}$ the last entry of which reveals the key d.

3 Signal Analysis

3.1 Experimental Setup

To measure the EM leakage from the smartphone, we used a Langer LF-R 400 near field probe (a 25mm loop probe, 100 kHz–50 MHz). We amplified the signal measured by the probe using a (customized) Mini-Circuits ZPUL-30P amplifier, providing 40 dB of gain. The output of the amplifier was then low-pass filtered at 5 MHz.

For digitizing the analog signal, we used one of two instruments. For the best robustness during initial characterization, as described below in this subsection, we used a National Instruments PCI-6115 data acquisition device, sampling at 10 Msample/sec with 12 bits of ADC resolution. For key extraction, described in Section 3.2, we used an Ettus N200 software defined radio device, with its LFRX daughterboard, sampling at 1 Msample/sec.

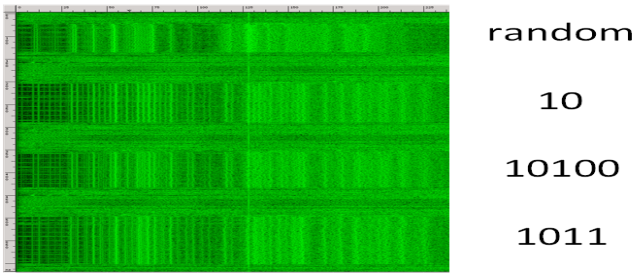


Figure 2: EM measurement (0.5 sec, 0–225 kHz) of four scalar-by-point multiplication operations using the NIST P-521 curve executed on an iPhone 3GS smartphone. The scalar was set to be either a random 521-digit number or the 521-digit number obtained by repeating the pattern written to the right. In all cases, the same curve point was used to perform the multiplication.

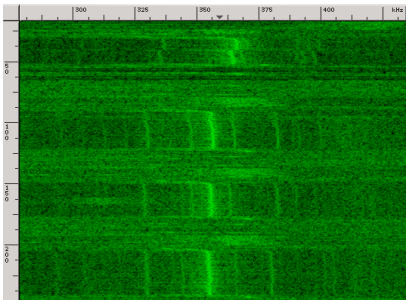
Note that in terms of bandwidth, the above is an overkill for our attacks, which exploit signals up to 200 kHz. Thus, similarly to [29], we can replace the probe and data acquisition device with much cheaper equipment, such as a sound card; this is discussed in Section 4.

Scalar-Dependent Leakage. Confirming the existence of scalar-dependent leakage from OpenSSL's scalar-by-point multiplication function, Figure 2 shows a spectrogram of the EM leakage obtained during four distinct signature operations, using the same point $\mathbb{P}$ and four different values of the scalar k. Notice that all of the four scalars can be easily distinguished by changes in vertical line pattern in their spectral signature. Such nonce-dependent leakage was observed on many target phones, of various models and manufactures (see Figure 3). This hints at the relationship between the time behavior of the observed leakage signal and the secret bits of the scalar k.

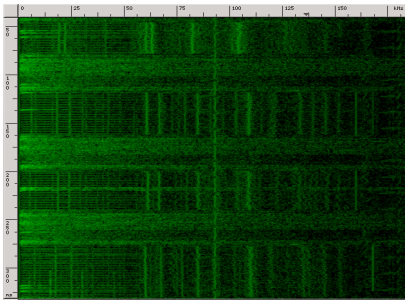
Triggering. In all the key-extraction attacks presented below, we simulated a completely passive attacker which does not interact with the target device. In particular (as mentioned in Section 1.2) we did not use any software-based or hardware-based triggering of the measurement setup (unlike [15]). Instead, we sampled continuously and relied on our signal processing to locate the leakage from the signing operation within the measurement trace (see details below). In order to conveniently attack multiple cryptographic libraries (with different API interfaces and running on different mobile operating systems), for each attacked library we wrote a small program that calls the ECDSA operation, and invoked it over the network.

3.2 Attacking OpenSSL ECDSA

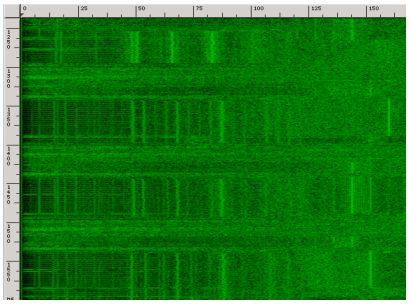
Signal Acquisition. We recorded the leakage of 5000 OpenSSL ECDSA signatures executed on an iPhone 3GS. For all of the recorded signatures, we used the `secp256k1` curve with the same randomly-generated secret key. We measured the iPhone's electromagnetic emanations during the signing operations using the setup described in Section 3.1 (with the Ettus N200 sampling at 1 Msample/sec).



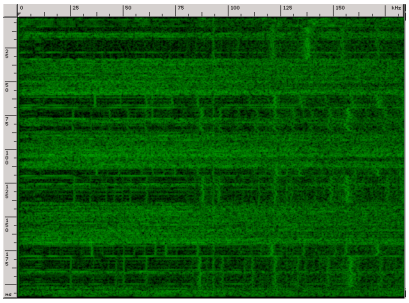
(a) Apple iPad Mini 2
$(0 - 400\,\text{kHz}, 0.2\,\text{msec})$.



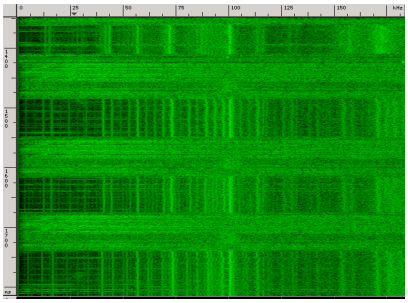
(b) Apple iPad 3rd Generation
$(0 - 200\,\text{kHz}, 0.3\,\text{msec})$.



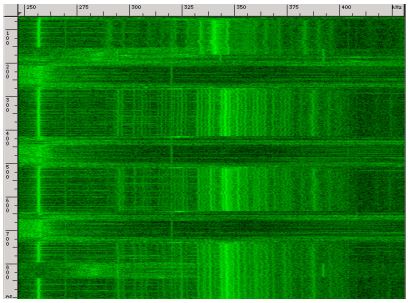
(c) Apple iPhone 4s
$(0 - 175\,\text{kHz}, 0.3\,\text{msec})$.



(d) Apple iPhone 5s
$(0 - 200\,\text{kHz}, 0.2\,\text{msec})$.



(e) Apple iPod Touch 4th Generation
$(0 - 200\,\text{kHz}, 0.2\,\text{msec})$.



(f) Sony Ericsson Xperia X10
$(250 - 500\,\text{kHz}, 0.8\,\text{msec})$.

Figure 3: EM measurement of four scalar-by-point multiplication operations using the NIST P-521 curve executed on various mobile devices. In each subfigure, the first multiplication used a random 521-digit scalar while the remaining three used the same repetitive 521-digit numbers used in Figure 2 (in the same order). Similarly to Figure 2, the same curve point was used to perform the multiplication.

We then stored the recorded traces, as well as the signed message produced by the ECDSA signing, for offline signal processing and cryptanalysis.

Examining Raw Traces. After digitizing, we applied a Finite Impulse Response (FIR) low-pass filter to suppress noise outside the 0–125 kHz band. The resulting signal can be seen in Figure 5 (top). Evidently, even after suppressing high frequency noise, one still cannot easily determine the locations of point addition and point doubling operations. In addition, the signal is periodically corrupted by strong disturbances caused by the operating system timer interrupts. Previous works ([29–31]) have mitigated the problem of low SNR and signal distortion by repeating each measurement several times and combining the results into a single clear aggregate trace. However, this is inapplicable to ECDSA: each signing operation uses a different nonce k, so the corresponding scalar-by-point multiplications $[k]\mathbb{G}$ results in different DA-sequences that cannot be directly combined.

Locating Signing Operations. In order to successfully execute our attack, we need to find the exact points in time where each signing operation ends. Unlike the concurrent work of [14] which assumes that these exact time points are leaked via the USB port, we assume the attacker has no leakage of such information and tackle the problem during our signal processing steps. For this purpose we utilize a distinct trace pattern occurring at the very end of each signing. This pattern is a natural product of the executed code (it is not an artificial trigger), but is very similar across different signing operations, for given software and hardware. After performing signal denoising (described next) we apply correlation-based detection to identify all instances where this distinct pattern occurs. We thus obtain the end points for most signing operations. We ignored some traces that ended in distorted patterns that did not correlate well.

Denoising Signal Traces. As mentioned above, an average-based denoising approach is not applicable to ECDSA since each signing operation uses a different nonce k. Instead, to increase the SNR of our traces, we add a preprocessing step (following the FIR filter) that performs Singular Spectrum Analysis (SSA). SSA can be used for blind source separation and denoising of single traces [34]. In the context of side channels, SSA was used in [53] to increase the success probability of various DPA-style attacks targeting embedded devices. The aim of the SSA procedure is to decompose a given time series $\{a_i\}_{i=1}^{N}$ into several distinct components, each with its own physical properties. The algorithm consists of three stages (see [34] for further details).

Step 1: Embedding. First, a window length $2 < L < N/2$ is chosen and used to construct a *trajectory matrix* of the input time series $\{a_i\}_{i=1}^{N}$. The trajectory matrix is comprised of K "lagged" copies of the series and is defined as follows (where A_i are column vectors and $K \doteq N - L + 1$):

$$\mathbf{A} = \begin{pmatrix} A_1 & A_2 & \dots & A_K \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_K \\ a_2 & a_3 & \dots & a_{K+1} \\ \vdots & \vdots & \ddots & \vdots \\ a_L & a_{L+1} & \dots & a_N \end{pmatrix}.$$

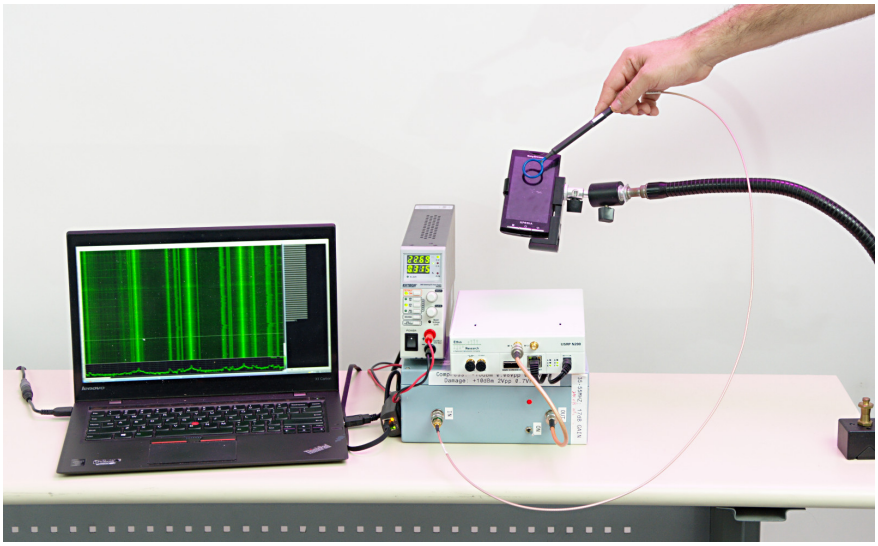


Figure 4: Our lab-grade setup attacking a Sony-Ericsson Xperia x10 phone. Left to right: analysis laptop, power supply, ZPUL 30P amplifier (gray box), Ettus N200 (white box), and phone being attacked using the Langer LF-R 400 probe (blue).

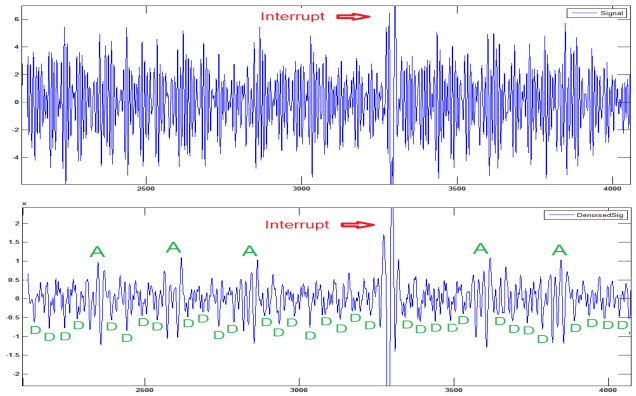


Figure 5: A recorded trace after filtering out high frequency noise (top), and the same trace after additionally applying SSA (bottom). Note the timer interrupt disturbing the measurement signal.

Notice $\mathbf{A}$ is also a *Hankle matrix* since all entries on its anti-diagonal are identical, i.e. for all $2 \leqslant i+j \leqslant N+1$ and $1 \leqslant n \leqslant N$ we get $\mathbf{A}_{ij} = a_n$ if $i+j = n+1$.

Step 2: Singular Value Decomposition. After obtaining the trajectory matrix, it is decomposed using a Singular Value Decomposition (SVD). An SVD of a matrix $\mathbf{A}$ is a well known decomposition and is defined by (for an L by K matrix with $L < K$):

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T = \begin{pmatrix} U_1 \\ \cdots \\ U_L \end{pmatrix}^T \begin{pmatrix} \sqrt{\lambda_1} & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{\lambda_L} & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ \vdots \\ V_K^T \end{pmatrix},$$

where λ_i are the eigenvalues of $\mathbf{A}\mathbf{A}^T$ in descending order and U_i are their corresponding eigenvectors. The vectors V_i are the eigenvectors of $\mathbf{A}^T\mathbf{A}$. For $1 \leqslant i \leqslant d$, the vectors V_i can be given by $V_i = \frac{\mathbf{A}^T U_i}{\sqrt{\lambda_i}}$, where d is the lowest eigenvalue such that $\lambda_d > 0$. The SVD of a matrix thus allows us to represent it as a sum of matrices: $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T = \sum_{i=1}^{d} \sqrt{\lambda_i} U_i V_i^T = \sum_{i=1}^{d} \sqrt{\lambda_i}\mathbf{X}_i$. The matrices $\mathbf{X}_i$ are called *projection matrices*, and their contributions to the original matrix $\mathbf{A}$ are proportional to $\sqrt{\lambda_i}$ (which are also called the *singular values* of $\mathbf{A}$).

Step 3: Reconstruction. Each matrix $\mathbf{X}_i$ can now be transformed back into a length N time series $\{x_n^i\}_{n=1}^N$ by averaging over the entries in its anti-diagonal. This process is also called *diagonal averaging* or *Hankelazation* [35]. Overall we obtain a decomposition of the original time series $\{a_n\}_{n=1}^N$ into a sum of d series: $\{a_n\}_{n=1}^N = \sum_{i=1}^{d} \{x_n^i\}_{n=1}^N$. A denoised time series can now be reconstructed by choosing a suitable subset of $m \leq d$ series from within the set $\{x_n^i\}_{i=1}^d$.

SSA Parameter Choice. The quality of the decomposition and denoising is highly dependent on the window size L and the choice of subset m. Empirically, we have found that good results are obtained when L is chosen to be shorter than both the DOUBLE and the ADD operations. At a sampling rate of 1M samples per second, the length of DOUBLE and ADD operations was 50 and 250 samples, respectively. We thus chose L to be 10 samples long. The reconstruction subset m was also chosen empirically. We found that a good result is achieved when one uses the components corresponding to the third, fourth and fifth highest singular values for reconstruction (regarding the rest of the components as noise). It is worth noting that SSA is often used in the literature to expose hidden periodic trends in noisy signals. In these cases it is recommended [34, 39] to choose L to be relatively large (larger than the longest suspected hidden period). However, in our case the DA sequence has no intrinsic periodicity, and larger L values seemed to be less suited for denoising.

Figure 5 depicts a signal trace after undergoing the SSA procedure. The DOUBLE and ADD sequence can now be

clearly seen. Note that the SSA procedure did not get rid of the interrupt induced disturbances, but since we only require a small number of DOUBLE operations taken from the end of the trace, this is usually not an issue. On the rare occasions where an interrupt was detected at the very end of the trace, the corresponding recording was simply discarded.

Locating Addition Operations In Time. Examining the denoised trace, we can now attempt to extract the DA-sequence. For this purpose we turn to the time-frequency domain. The middle of Figure 6 depicts the spectrogram of a denoised trace, where the frequency band containing most of the energy of addition operations becomes especially clear (see Figure 6 (middle)). Summing over the spectrogram's energy we receive a trace marking the locations of addition operations in time . In order to increase the detection accuracy, we enhance this trace by multiplying it with its own derivative with respect to time. This way we are able to enhance high amplitude peaks that also rise sharply, and attenuate other peaks. Further smoothing produces the signal depicted in Figure 6 (bottom), where the peaks marking the locations of ADD operations can be detected with high fidelity.

Extracting the Partial DA-Sequence. Having found a way to detect addition operations with sufficient fidelity, we can now attempt to locate the position of the very last addition operation in the signal. We do so by first finding the point in each trace where the signing operation ends. This point can be reliably found in many traces since the signal pattern that immediately follows the signing operation is consistently similar across many traces. We can use this pattern as a template to reliably locate it in other traces using correlation, discarding traces that do not correlate well with the chosen template. By measuring the distance between the estimated template location in each trace and the very last addition operation detected in the signature (found using the methods described in the previous subsection), we can determine the number of DOUBLE operations that occurred at the very end of each signing operation, thus acquiring the DA information necessary for key extraction.

Signal Analysis Performance. Applying our attack to a randomly-generated ECDSA secp256k1 OpenSSL key, we measured the EM emanations during 5000 signatures on an Apple iPhone 3GS smartphone, each signature lasting 0.1 sec. Applying our signal processing to the 5000 traces we collected, we were able to detect the end time of the signing operation in 1278 traces. Out of these, 114 traces were identified as having their DA-sequence terminate with at least three elliptic curve DOUBLE operations; 3 of these were false positives (as discovered in retrospect; the attack code did not use this information).

Lattice Reduction and Key Extraction. Using the above 114 traces, we randomly selected 85 traces (this number was set empirically, to obtain high success probability in the next step) and applied the lattice attack of [16] using the fplll [7] implementation of the BKZ algorithm with block size $\beta = 30$. Unfortunately, whenever the selected traces happen to include some of 3 erroneous ones, the BKZ algorithm fails to recover the signing key, causing the key-extraction attempt to fail. Therefore, we repeated the procedure of randomly selecting 85 traces and applying the lattice attack 30 times. Since each lattice reduction attempt does not depend on others, we performed these repetitions in a parallel manner on separate cores. Across the 30 parallel attempts,

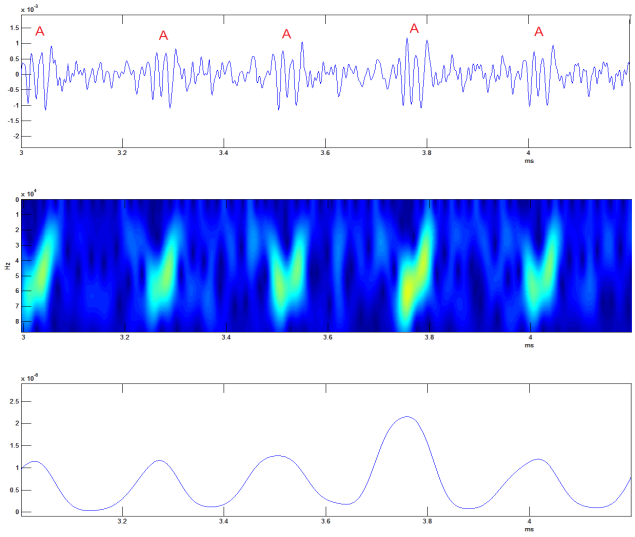


Figure 6: (top) A denoised trace with add operations marked. (middle) Zoomed-in view of the spectrogram of the trace. The energy of add operations is clearly visible. (bottom) Energy as a function of time of the visible part of the spectrogram (after enhancement and smoothing). Peaks approximate the location of add operations.

the secret key was successfully recovered in 2. The signal processing and lattice reductions took two hours on a desktop PC (3.4 GHz, 6 cores, 64 GB RAM), leading to complete extraction of the ECDSA signing key. Notice that all the 30 repetitions of the lattice reduction step were done offline on the *same* data base of analog traces. Thus, even a single successful lattice reduction leads to a successful key-recovery attack.

3.3 Other ECSDA Implementations

Attacking OpenSSL on Android Devices. Key-dependent leakage, similar to Figure 2, was also observed on various Android phones. See Figure 3. We thus conjecture that similar attacks can be mounted on these devices as well. Demonstrating the feasibility of such a result on an Android device, Figure 7 shows the extraction of a sequence of elliptic curve DOUBLE and ADD operations from a Sony-Ericsson Xperia X10 smartphone. The signal in the figure is the result of digital FM demodulation and filtering.

Attacking CoreBitcoin. Demonstrating the possibility of Bitcoin theft via side-channel from iOS devices, we have mounted a successful key extraction attack on CoreBitcoin's low s-value ECDSA implementation running on iOS. We recorded the leakage of 5000 ECDSA secp256k1 signatures executed on an Apple iPhone 3GS smartphone. Out of these 5000 traces, 1940 were discarded due to measurement noise. Out of the remaining 3060 traces, 110 were identified as having their DA sequence terminate in at least four elliptic curve DOUBLE operations with one of these being a false positive (again, discovered in retrospect). Next, we randomly chose 85 out of the 110 available traces and applied the lattice attack described in Section 2.3. Repeating the attack 20 times (each time choosing a random subset containing 85 traces) resulted in a successful key extraction in 4 out of the 20 attempts. Similar to Section 3.2, all the 20 repetitions of the

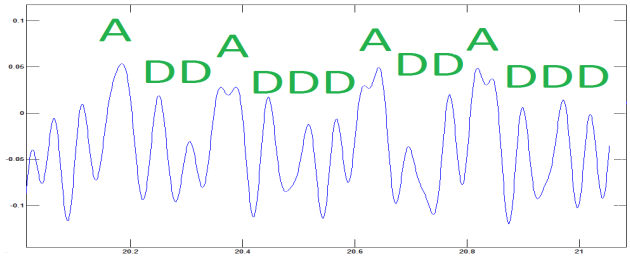


Figure 7: Sequence of double and add operations extracted during a secp256k1 scalar by point multiplication operation executed on an Sony-Ericsson Xperia X10 smartphone. In this experiment we replaced the wNAF representation of k with the 256-digit string obtained by repeating the pattern 10100.

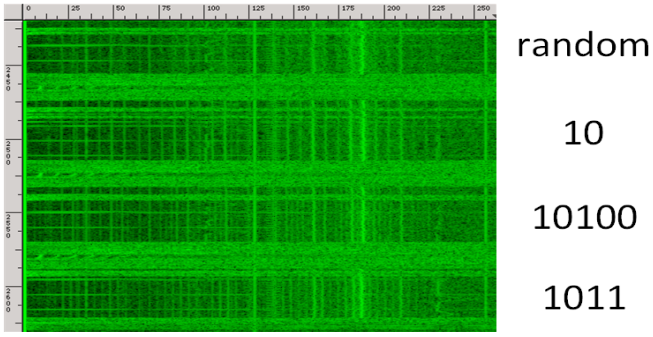


Figure 8: EM measurement (0.2 sec, 0–250 kHz) of four scalar-by-point multiplication operations using the NIST P-521 curve executed on an iPhone 3GS smartphone running Apple's CommonCrypto library. As in Figure 2, the scalar was set to be either a random 521-digit number or a the 521-digit number obtained by repeating the pattern written to the right. In all cases, the same curve point was used to perform the multiplication.

lattice reduction step were done offline on the *same* data base of analog traces. Thus, even a single successful lattice reduction leads to a successful key-recovery attack.

Attacking Apple's CommonCrypto ECDSA Implementation. The ECDSA implementation of Apple's CommonCrypto library performs the elliptic curve scalar-by-point multiplication operation using Algorithm 1 with $w = 1$. Figure 8 shows scalar-dependent leakage, similar to Figure 2, obtained by measuring an iPhone 3GS when invoking the elliptic curve multiplication operation as implemented in Apple's CommonCrypto library.

4 Cheap Attacks

While the results of Section 3 clearly demonstrate the possibility of key extraction via the electromagnetic channel using expensive lab equipment, the low bandwidth nature of our attacks allows for key extraction using a much cheaper experimental setup via both the electromagnetic channel and the power channel.

EM Probe. For the EM channel we improvised a probe by scavenging a coil from a Qi wireless charging receiver module ($2 on eBay). See Figure 1.

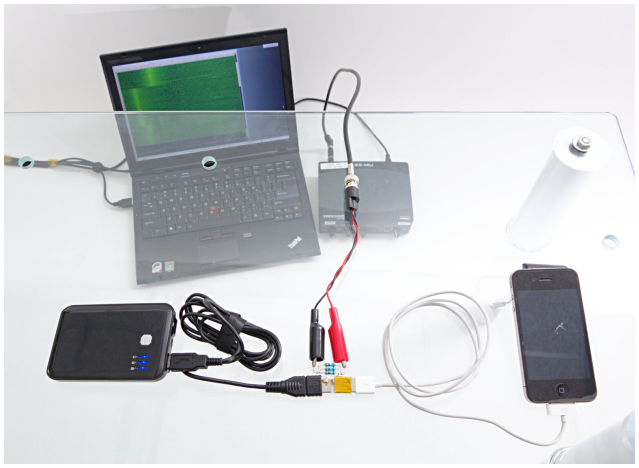


Figure 9: Mounting a cheap power analysis attack on an iPhone 4 through its charging port. This setup includes a portable battery (bottom, left), a power monitoring probe (bottom, middle) and an iPhone 4 (bottom, right). The power probe is then connected to the Tracker Pre sound card (top, middle) and the attacker's laptop (top, left).

Power Probe. To monitor the phone's current draw, we built a simple USB pass-through adapter, with a 0.33Ω resistance in series with the ground line. We then connected the phone to a portable USB battery pack through the pass-through adapter, and measured the voltage over the resistor; see Figure 9.

Digitizer. We connected the improvised EM and power probes into the microphone input of a Creative Tracker Pre sound card ($50, eBay). This card acts both as an amplifier (60dB gain) and as a digitizer (192 Ksample/sec).[4]

Attack Scenario. Small loops of wire acting as EM probes can be easily concealed inside various objects (such as tabletops, phone cases (especially those containing an extra battery), or even food items [29]). See Figure 1. Monitoring the phone's power consumption can be easily done by augmenting an aftermarket charger, external battery or battery case with the requisite equipment. In this context, phone cases which contain an additional battery (and therefore are connected to the phone's charging port) are especially dangerous since these can be augmented to monitor both channels simultaneously, thus obtaining a potentially cleaner signal. We leave this for future work.

Scalar-Dependent Leakage. We measured the EM leakage of an iPhone 4 using our improvised EM probe connected to the Tracker Pre sound card and concealed beneath a glass tabletop (see Figure 10). Similarly to Figure 3, Figure 10 (right) presents a spectrogram of five distinct signature operations, using the same point $\mathbb{P}$ and five different values of the scalar k. Notice that even though the equipment used to generate Figure 10 is much simpler (and cheaper) than the lab-grade equipment used in Figure 3, the five different

[4] Alternatively, one could use an inexpensive USB oscilloscope. However, these are optimized for bandwidth (at the expense of SNR), and would require an additional amplifier. Sound cards typically offer higher SNR and a built-in amplifier, and while their bandwidth is much lower, it suffices for our attacks.

scalars can be easily distinguished from their spectral signature. Similar results were obtained using the power probe as well (see Figure 10 (left)).

Extracting the DA-Sequence. After observing the scalar dependent leakage using our improvised probes and the Tracker Pre sound card, we proceeded to extract the DA-sequence which is required for our attack. Applying our signal processing techniques on an iPhone 3GS running OpenSSL secp256k1 signature operations, Figure 11 depicts the results of extracting the DA-sequence using EM leakage from a single signature. Notice that the individual DOUBLE and ADD operations can clearly be seen. Repeating this process for about 5000 signatures should result in a complete key recovery.

5 Conclusions

In this paper we have demonstrated that despite its speed and randomization, ECDSA signatures on mobile devices are vulnerable to physical key extraction attacks. Moreover, the attacks can be mounted cheaply and non-intrusively. Our attack exploits the differences between point addition and point doubling to recover the DA-sequence. Two approaches can be used to protect an implementation from side-channels. The value of the nonce can be decoupled from the DA-sequence using *blinding*. Alternatively, the implementation can be modified to always use the same DA-sequence, irrespective of the value of the nonce.

Nonce Splitting. Clavier and Joye [23] suggest expressing the nonce k as $k = k_1 + k_2$, where k_1 is random and then compute $[k_1]\mathbb{G} + [k_2]\mathbb{G}$ using a multi-exponentiation algorithm [43]. However, this approach leaks the least significant bits of k_1 and k_2, as well as long, overlapping sequences of repeating bits in k_1 and k_2. Splitting the nonce more, i.e., expressing $k = k_1 + k_2 + k_3 + \cdots$, such that all the terms but one are chosen at random, can reduce the probability of overlaps. However, this approach still leaks the least significant bit of k and as [12] show, leaks of one bit can be exploitable.

Nonce Blinding. Coron [24] suggests blinding the nonce by choosing a random c and calculating $[k]\mathbb{G} + [cn]\mathbb{G}$ where n is the group order. Ciet and Joye [22] note that for groups of order close to a power of two, this still leaks information about the high bits of k, and Van de Pol et al. [51] show how to exploit such leaks. Combining the two approaches, i.e., calculating $\sum[k_i]G$ for random k_i's and c such that $\sum k_i = k + cn$, protects from both types of leaks.

Constant-time Implementations. Constant-time implementations mitigate many side-channel leaks by ensuring a fixed execution path that does not depend on secret data, to prevent timing attacks [21]. Additionally, a *constant memory access pattern* is desired to avoid cache-based attacks [17, 49, 50], as well as cache-induced differences in timing and electromagnetic behavior. For, EC scalar-by-point multiplication, the scalar can be represented in a regular way such that the DA-sequence does not depend on the nonce [36, 44]; Moller [44] notes that these encodings may leak information when a point is added to itself, yet with a random scalar, as is the case in ECDSA, the probability of this leak is negligible. A constant-time implementation for some elliptic curves, on some 64-bit platforms, is included in OpenSSL [37]. For Bitcoin's secp256k1 curve, the libsecp256k1 [57] implementation offers a constant-time, constant-memory-access implementation of ECDSA signing.

Future Work. While this work clearly demonstrates the vulnerability of multiple implementations of ECDSA signatures running on mobile devices to cheap low-bandwidth key extraction attacks, much works remains to be done. The vulnerability of other ECDSA implementations, as well as general cryptographic code running on mobile devices has received much research attention. Improving the signal quality, thereby increasing the attack range and reducing the number of required signatures is an intriguing open problem. To that end, we note that the more advanced lattice techniques of [8, 51] are of potential use in order to reduce the number of signatures. However, our signal is too corrupted (due to interrupts) making these techniques inapplicable without significant improvements in signal processing techniques.

Acknowledgments

Daniel Genkin, Lev Pachmanov, Itamar Pipman and Eran Tromer are members of the Check Point Institute for Information Security. This work was done in part while Eran Tromer was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

This work was supported by the Blavatnik Interdisciplinary Cyber Research Center; by a Google Faculty Research Award; by the Israeli Centers of Research Excellence I-CORE program (center 4/11); by the Leona M. & Harry B. Helmsley Charitable Trust; and by NATO's Public Diplomacy Division in the Framework of "Science for Peace".

6 References

[1] Bitcoin Core. URL: https://bitcoin.org/en/bitcoin-core/.

[2] CoreBitcoin Library. URL: https://github.com/oleganza/CoreBitcoin.

[3] Working with micropayment channels. URL: https://bitcoinj.github.io/working-with-micropayments.

[4] SPA/SEMA vulnerabilities of popular RSA-CRT sliding window implementations, 2012. presented at CHES 2012 rump session. URL: https://www.cosic.esat.kuleuven.be/ches2012/ches_rump/rs5.pdf.

[5] D. Aboulkassimi, M. Agoyan, L. Freund, J. Fournier, B. Robisson, and A. Tria. Electromagnetic analysis (EMA) of software AES on Java mobile phones. In *WIFS 2011*, pages 1–6. IEEE, 2011.

[6] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel(s). In *CHES 2002*, pages 29–45. Springer, 2002.

[7] M. Albrecht, S. Bai, D. Cadé, X. Pujol, and D. Stehlé. fplll-4.0, a floating-point LLL implementation. URL: http://perso.ens-lyon.fr/damien.stehle.

[8] T. Allan, B. B. Brumley, K. E. Falkner, J. van de Pol, and Y. Yarom. Amplifying side channels through performance degradation. Cryptology ePrint Archive, Report 2015/1141, 2015. http://ia.cr/2015/1141.

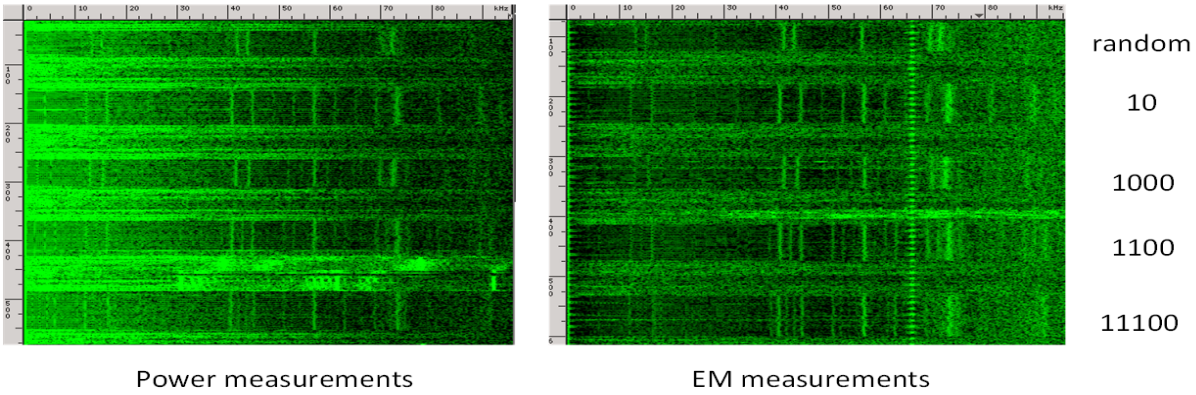


Figure 10: Power (left) and EM (right) measurement (0.2 sec, 0–96 kHz) of five scalar-by-point multiplication operations using the NIST P-521 curve executed on an iPhone 4 smartphone running OpenSSL. As in Figure 2, the scalar was set to be either a random 521-digit number or a the 521-digit number obtained by repeating the pattern written to the right. In all cases, the same curve point was used to perform the multiplication.

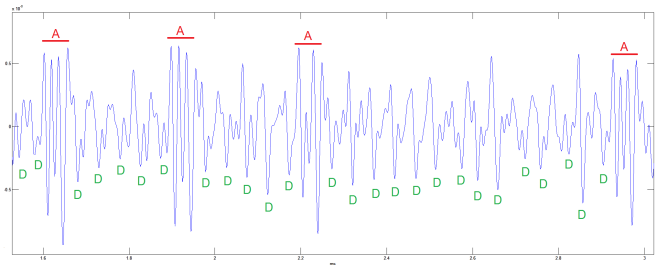


Figure 11: Extracted DA-sequence obtained from iPhone 3GS during an OpenSSL `secp256k1` signature. The leakage was measured using the Tracker Pre sound card and the improvised EM probe. The double and add operations can clearly be seen.

[9] R. J. Anderson. *Security Engineering — A Guide to Building Dependable Distributed Systems (2nd ed.)*. Wiley, 2008.

[10] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. On the malleability of Bitcoin transactions. In *FC 2015*, pages 1–18, 2015.

[11] Apple Inc. Cryptographic libraries. URL: https://developer.apple.com/cryptography/.

[12] D. F. Aranha, P.-A. Fouque, B. Gérard, J.-G. Kammerer, M. Tibouchi, and J.-C. Zapalowicz. GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In *ASIACRYPT 2014, Part I*, pages 262–281. Springer, 2014.

[13] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede. DPA, bitslicing and masking at 1 GHz. In *CHES 2015*, pages 599–619. Springer, 2015.

[14] P. Belgarric, P.-A. Fouque, G. Macario-Rat, and M. Tibouchi. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. In *CT-RSA 2016*. Springer, 2016. To Appear.

[15] P. Belgarric, P.-A. Fouque, G. Macario-Rat, and M. Tibouchi. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. Cryptology ePrint Archive, Report 2016/231, 2016. http://ia.cr/2016/231.

[16] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom. "Ooh aah... just a little bit" : A small amount of side channel can go a long way. In *CHES 2014*, pages 75–92, 2014.

[17] D. J. Bernstein. Cache-timing attacks on AES. http://cr.yp.to/papers.html#cachetiming, 2005.

[18] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *CRYPTO 1996*, pages 129–142, Santa Barbara, CA, US, Aug. 1996.

[19] B. B. Brumley and R. M. Hakala. Cache-timing template attacks. In *ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 667–684. Springer, 2009.

[20] B. B. Brumley and N. Tuveri. Remote timing attacks are still practical. In *ESORICS 2011*, pages 355–371. Springer, 2011.

[21] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[22] M. Ciet and M. Joye. (virtually) free randomization techniques for elliptic curve cryptography. In *ICICS 2003*, pages 348–359. Springer, 2003.

[23] C. Clavier and M. Joye. Universal exponentiation algorithm. In *CHES 2001*, pages 300–308. Springer, 2001.

[24] J. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES 2002*, pages 292–302, 1999.

[25] J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-art of secure ECC implementations: A survey on known side-channel attacks and countermeasures. In *HOST 2010*, pages 76–87, 2010.

[26] J. Fan and I. Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, pages 265–282, 2012.

[27] J. L. Galea, E. D. Mulder, D. Page, and M. Tunstall. Soc it to EM: electromagnetic side-channel attacks on

a complex system-on-chip. In *CHES 2015*, pages 620–640. Springer, 2015.

[28] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: concrete results. In *CHES 2001*, pages 251–261. Springer, 2001.

[29] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *CHES 2015*, pages 207–228, 2015. Extended version: Cryptology ePrint Archive, Report 2015/170.

[30] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer. ECDH key-extraction via low-bandwidth electromagnetic attacks on PCs. In *CT-RSA 2016*, pages 219–235, 2016.

[31] D. Genkin, I. Pipman, and E. Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. In *CHES 2014*, pages 242–260. Springer, 2014. Extended version: Cryptology ePrint Archive, Report 2014/626.

[32] D. Genkin, A. Shamir, and E. Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO 2014*, pages 444–461 (vol. 1). Springer, 2014. Extended version: Cryptology ePrint Archive, Report 2013/857.

[33] G. Goller and G. Sigl. Side channel attacks on smartphones and embedded devices using standard radio equipment. In *COSADE 2015*, pages 255–270. Springer, 2015.

[34] N. Golyandina and A. Zhigljavsky. *Singular Spectrum Analysis for Time Series*. Springer, 2013.

[35] H. Hassani. Singular spectrum analysis: Methodology and comparison. *J. of Data Science*, (5):239–257, 2007.

[36] M. Joye and M. Tunstall. Exponent recoding and regular exponentiation algorithms. In *AFRICACRYPT 2009*, pages 334–349. Springer, 2009.

[37] E. Käsper. Fast elliptic curve cryptography in OpenSSL. In *FC 2011*, pages 27–39. Springer, 2012.

[38] G. Kenworthy and P. Rohatgi. Mobile device security: The case for side channel resistance. In *MoST 2012*, 2012.

[39] M. A. R. Khan and D. Poskitt. Window Length Selection and Signal-Noise Separation and Reconstruction in Singular Spectrum Analysis. Monash Econometrics and Business Statistics Working Papers 23/11, Monash University, Department of Econometrics and Business Statistics, 2011. URL: https://ideas.repec.org/p/msh/ebswps/2011-23.html.

[40] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[41] M. Lipp, D. Gruss, R. Spreitzer, and S. Mangard. ARMageddon: Last-level cache attacks on mobile devices. *CoRR*, abs/1511.04897, 2015. http://arxiv.org/abs/1511.04897.

[42] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks — Revealing the Secrets of Smart Cards*. Springer, 2007.

[43] B. Möller. Algorithms for multi-exponentiation. In *SCA 2001*, pages 165–180. Springer, 2001.

[44] B. Möller. Securing elliptic curve point multiplication against side-channel attacks. In *ISC 2001*, pages 324–334. Springer, 2001.

[45] Y. Nakano, Y. Souissi, R. Nguyen, L. Sauvage, J. Danger, S. Guilley, S. Kiyomoto, and Y. Miyake. A pre-processing composition for secret key recovery on Android smartphone. In *WISTP 2014*, pages 76–91. Springer, 2014.

[46] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, 2013.

[47] P. Q. Nguyen. Lattice reduction algorithms: Theory and practice. In *Eurocrypt 2011*, pages 2–6, Tallinn, Estonia, May 2011.

[48] P. Q. Nguyen and I. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.

[49] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA 2006*, pages 1–20. Springer, 2006.

[50] C. Percival. Cache missing for fun and profit. Presented at BSDCan. http://www.daemonology.net/hyperthreading-considered-harmful, 2005.

[51] J. van de Pol, N. P. Smart, and Y. Yarom. Just a little bit more. In *CT-RSA 2015*, pages 3–21, 2015.

[52] T. Pornin. Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA). RFC 6979, 2013.

[53] S. M. D. Pozo and F. Standaert. Blind source separation from single measurements using singular spectrum analysis. In *CHES 2015*, pages 42–59. Springer, 2015.

[54] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *E-smart 2001*, pages 200–210, 2001.

[55] G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.

[56] C. Smith. What are micropayments and how does Bitcoin enable them?, 2015. URL: https://coincenter.org/2015/06/what-are-micropayments-and-how-does-bitcoin-enable-them/.

[57] P. Wuille. libsecp256k1. URL: https://github.com/bitcoin/secp256k1.

[58] P. Wuille. Dealing with malleability, 2014. URL: https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki.

[59] A. Zajic and M. Prvulovic. Experimental demonstration of electromagnetic information leakage from modern processor-memory systems. *EMC 2014*, 56(4):885–893, 2014.

APPENDIX

A Current Status of Targeted Software

This appendix reviews the vulnerability status of the targeted software, at the time of writing. See also Table 1.

OpenSSL 1.0.x branch. We have conducted most of our experiments on OpenSSL version 1.0.1m which was the latest version at the time of conducting this research. For ARM processors, all curves over prime fields in the current versions of OpenSSL (1.0.1r and 1.0.2f) use the same underlying el-

liptic curve implementation and thus appear vulnerable to attacks; while we did not attempt key extraction, scalar-dependent leakage (similar to Figure 2) was empirically observed from these OpenSSL versions as well. Upon contacting OpenSSL we were notified that "hardware side-channel attacks are not in OpenSSL's threat model", so no updates are planned to OpenSSL 1.0.x to mitigate our attacks. Note that OpenSSL 1.0.2 will be supported until the year 2020,

OpenSSL 1.1.x branch. OpenSSL 1.1.0 pre-release 3 includes an ARM-specific constant-time implementations of the NIST P-256 curve, which is unlikely to be vulnerable to similar attacks. All other curves over prime fields, including the secp256k1 curve, still use the vulnerable wNAF implementation.

iOS 7.1.2–8.3 CommonCrypto. The ECDSA implementation in the CommonCrypto framework of iOS 7.1.2 appears vulnerable, since it exhibits scalar-dependent leakage (see Section 3.3). Reverse-engineering the code in iOS 8.3 reveals that it uses the same vulnerable implementation (wNAF with $w = 1$).

iOS 9.x CommonCrypto. Starting with iOS 9, CommonCrypto uses a new EC implementation, including side-channel mitigation techniques such as operand-independent control flow and Montgomery-ladder multiplication.[5] Our current attacks are not applicable to this new EC implementation, and we have no evidence that it is vulnerable.

CoreBitcoin. CoreBitcoin [2] (not to be confused with Bitcoin core, below) is currently vulnerable, as discussed in Section 3.3. In response. the CoreBitcoin developers expressed their intention to switch to the libsecp256k1 library [57] in the future. This library offers a constant-time, constant-memory-access implementation of ECDSA signing, and we have no evidence that it is vulnerable.

Bitcoin Core. The Bitcoin core code [1] (not to be confused with the CoreBitcoin, above) has already transitioned to the libsecp256k1 library [57] for ECDSA signing, starting from version v0.10.0 (released in February 2015).[6] This library offers a constant-time, constant-memory-access implementation of ECDSA signing, and we have no evidence that it is vulnerable.

[5]The relevant code is in the corecrypto library version 337 [11]. The Apple Product Security confirmed that this is, essentially, the implementation in all iOS 9.x versions, as well as OS X 10.11, and that 77% of the iOS installations are iOS 9.x, as of February 2016.

[6] Bitcoin Git commit `fda3fed18a` added support for libsecp256k1 ECDSA signing, and commit `dffb8f81b8` removed support for OpenSSL ECDSA signing.